PHP

БАЗОВЫЙ УРОВЕНЬ

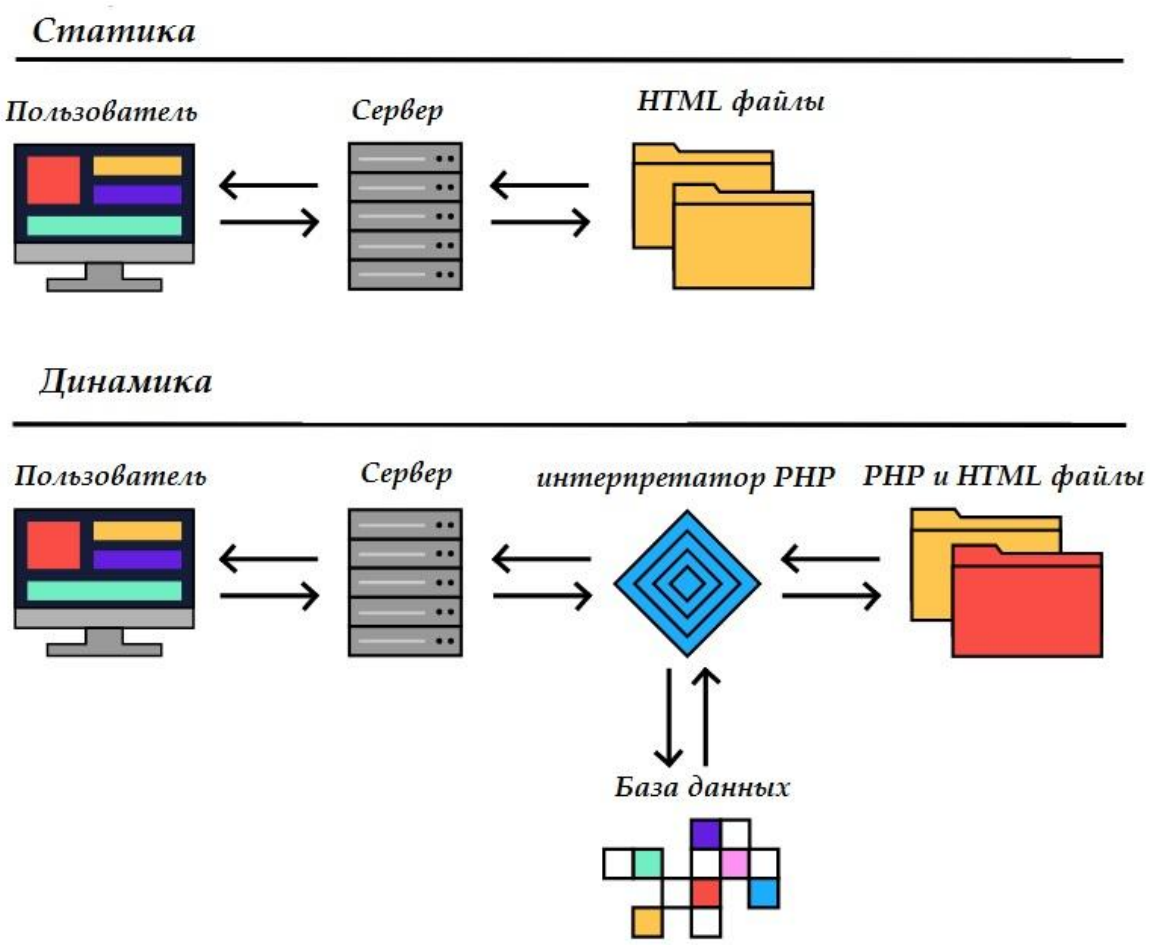
СОДЕРЖАНИЕ

ВВЕДЕНИЕ

- PHP является одним из самых популярных языков веб-разработки, на нем написаны более 77% сайтов в Интернете; сам язык продолжает активно развиваться — в ноябре 2022 года была представлена версия 8.2

- PHP применялся при разработке крупных сайтов (Facebook, Wikipedia), но в основном используется для небольших и средних коммерческих проектов; также на PHP написаны множество популярных CMS и платформ электронной коммерции (WordPress, Joomla, Magento, PrestaShop, OpenCart)

- хорошее знание PHP позволяет веб-разработчику быстро найти работу (спрос на рынке труда на этот язык довольно высок)



- является полноценным языком программирования, но как правило используется следующим образом: на сервер (упрощенно — компьютер в сети Интернет, на котором расположены сайты) от посетителя (клиента) приходит запрос на получение данных

некого ресурса (файла) — файл может быть отдан в оригинальном виде, а может перед отдачей быть обработан некой серверной технологией, чтобы разные клиенты могли видеть разный результат запроса — PHP является именно такой серверной технологией, которая позволяет делать содержимое сайтов более динамичным

- имеет относительно низкий порог входа — начать программировать на PHP относительно просто, а огромное количество бесплатных материалов в Интернете позволяет найти ответ на любой вопрос

- PHP обладает отличным дополнительным инструментарием и развитой экосистемой — фреймворки Symfony и Laravel многократно ускоряют разработку, а менеджер зависимостей Composer позволяет использовать тысячи уже готовых решений от других программистов

- широкое распространение, низкий порог входа (что снижает средний профессиональный уровень разработчиков) и ориентация на небольшие приложения не позволяет причислить PHP к самым высокооплачиваемым языкам программирования

- в последнее время PHP начал испытывать все большую конкуренцию со стороны других технологий (Node.js, Python, Go), однако прогноз на ближайшие 10 лет можно назвать благоприятным

- несмотря на активное развитие, PHP по-прежнему имеет ряд серьезных недостатков, связанных с обратной совместимостью, типизацией и выполнением кода в многопоточной среде

БАЗОВЫЙ СИНТАКСИС

РАЗМЕЩЕНИЕ PHP КОДА

- программа, написанная на языке PHP, может находиться в файле с абсолютно любым расширением (это зависит от настроек сервера), однако обычно используется расширение .php

- непосредственно php программой считается только тот текст, который расположен между тэгами <?php и ?> (если в файле php.ini применить настройку short_open_tag = On, то можно использовать короткие тэги <? и ?>, но этот подход является устаревшим и не используется в современной разработке), весь остальной текст в файле будет выводиться как есть (без обработки PHP интерпретатором):

```
<!DOCTYPE html>
<html>
     <head>
          <meta charset="utf-8">
          <title>My Site</title>
     </head>
     <body>
          <div>
              Это будет проигнорировано PHP.</div>
          <div>
          <?php
               // ... PHP код
          ?>
          </div>
          <div>
              Это также будет проигнорировано PHP.
          </div>
          <div>
          <?php
               // ... PHP код еще раз
          ?>
          </div>
     </body>
</html>
```

- Программа на PHP может располагаться как на нескольких строках, так и на одной единственной строке:

```
<?php
    // ... PHP код на 1 строке
    // ... PHP код на 2 строке
?>
```

```
<?php // ... PHP код на одной единственной строке ?>
```

- Если в файле находится только PHP код или после PHP кода в файле больше ничего нет — закрывающий тэг ?> можно не писать:

```
<?php
    // ... PHP код без закрывающего тэга
```

ПЕРЕМЕННЫЕ И КОНСТАНТЫ

- переменные являются основными элементами, которые используются в PHP выражениях — переменные могут хранить разные значения и взаимодействовать друг с другом

- имена переменных начинаются со знака доллара ($), после знака доллара должна идти буква или символ подчеркивания, а последующие символы в имени переменной могут быть буквами, цифрами или символом подчеркивания в любом количестве

- имена переменных чувствительны к регистру

- операции с переменными являются выражениями, а выражения в PHP завершаются точкой с запятой (;):

```php
<?php

$x = 42; // объявление числовой переменной

$x = $x - 2; // отнимем от переменной $x значение 2

$y = 'Hello'; // объявление строковой переменной
```

- иногда в программировании необходимо использовать переменную, значение которой нельзя менять (например, настройку конфигурации) - такие переменные называются константами, их объявление имеет особый синтаксис, а попытка переопределения приведет к аварийному завершению программы:

```php
<?php

const DAYS_IN_WEEK = 7; // объявление константы

define('MONTHS_IN_YEAR', 12); /* устаревший способ объявления константы */
```

ВЫВОД ПЕРЕМЕННЫХ И КОНСТАНТ НА ПЕЧАТЬ

- само по себе объявление переменных не отображается в браузере или командной строке — нам необходимо явно дать команду вывода на печать — это делается с помощью команды echo (также существует схожая функция print, но она используется гораздо реже):

```
<?php

$name = 'John'; // объявление переменной

echo $name; // вывод переменной на печать

echo 2 + 2; // выведется 4

const NUMBERS_OF_SEASONS = 4; // объявление константы

echo NUMBERS_OF_SEASONS; // вывод константы на печать
```

- иногда в html коде необходимо просто отобразить значение одной php переменной — для этого существует особая конструкция <?= ?> (эта запись не имеет никакого отношения к коротким тэгам):

```
<?php $season = 'autumn'; ?>

<div><?= $season ?></div>
<!-- нижняя запись будет идентична верхней -->
<div><?php echo $season; ?></div>
```

КОММЕНТАРИИ

- очень часто PHP код приходится комментировать - чтобы в будущем можно было бы быстро разбираться в той проблеме, которую этот код решает

- комментарии никак не влияют на исполнение кода — интерпретатор PHP их просто игнорирует

- существует 2 типа комментариев — однострочные и многострочные

- однострочные комментарии могут начинаться с // или с # - после этих символов любой текст до конца строки будет проигнорирован PHP интерпретатором:

```
<?php

// echo 'ничего не будет напечатано';
$x = 1; // ничего не выведется после объявления переменной
# в этом случае также ничего не будет выведено
```

- многострочные комментарии начинаются с /* и заканчиваются */ - весь код между этими символами будет проигнорирован (сколько бы строк это не занимало):

```
<?php

/*
     этот текст никогда
     не выведется на печать
*/
```

ОТОБРАЖЕНИЕ ОШИБОК

- при разработке программ часто случается допустить ошибку — PHP позволяет включать или выключать отображение ошибок, а также переключать отображение разных типов ошибок

- чтобы включить отображение ошибок, в файле php.ini надо применить настройку display_errors = On или вставить в PHP код следующую конструкцию:

```
<?php

ini_set('display_errors', 1);
```

- для настройки отображения уровней ошибок используется функция error_reporting:

```
<?php

error_reporting(E_ALL); // разрешает показывать все ошибки

error_reporting(E_WARNING); // только некритические ошибки

error_reporting(E_DEPRECATED); /* показывает ошибки,
связанные с устаревшими конструкциями php */
```

ТИПЫ ДАННЫХ 1: ЦЕЛОЕ ЧИСЛО (INTEGER)

ОБЪЯВЛЕНИЕ

```php
<?php

$int1 = 100; // десятичное целое число

$int2 = -25; // отрицательное число

$int3 = 0o123; /* восьмеричное число (1 * 8^2 + 2 * 8^1 + 3 * 8^0 = 83) */

$int4 = 0x3E8; /* шестнадцатеричное число (3 * 16^2 + 14 * 16^1 + 8 * 16^0 = 1000) */

$int5 = 0b101; /* двоичное число (1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 5) */

$int6 = 1_000_000; // удобная запись больших чисел (1000000)
```

ОПЕРАЦИИ

```php
<?php

$int = 10;

$int++; /* постинкремент - возвращает значение и увеличивает
на 1 (11) */

$int--; /* постдекремент - возвращает значение и уменьшает на
1 (10) */

++$int; /* преинкремент увеличивает на 1 и возвращает
значение (11) */

--$int; /* предекремент - уменьшает на 1 и возвращает
значение (10) */

$a = -$int; // отрицание (-10)

$b = $a + 20; // сложение (10)

$c = $b - 140; // вычитание (-130)

$d = 100 * 2; // умножение (200)

$e = 100 / 2; // деление (50)

$f = 17 / 3; /* если деление нацело невозможно, то результат
преобразуется в дробное число (5.6666666666667) */

$g = 100 % 3; // взятие целочисленного остатка (1)

$h = 10 ** 3; // возведение в степень (1000)
```

ОГРАНИЧЕНИЯ

- константа PHP_INT_MAX позволяет узнать максимальное значение целого числа

- константа PHP_INT_MIN позволяет узнать минимальное значение целого числа

- при выходе за пределы максимального значения целое число будет приведено к дробному числу с потерей точности, например, если PHP_INT_MAX эквивалентно 9223372036854775807, то PHP_INT_MAX + 1 эквивалентно 9223372036854800000

ТИПЫ ДАННЫХ 2: ЧИСЛО С ПЛАВАЮЩЕЙ ЗАПЯТОЙ (FLOAT)

ОБЪЯВЛЕНИЕ

```php
<?php

$float1 = 6.25; // число с плавающей запятой

$float2 = -17.5; // отрицательное число

$float3 = 22E+10; /* запись больших чисел (перенос запятой на 10 цифр вправо - 220000000000) */

$float4 = 15E-7; /* запись малых чисел (перенос запятой на 7 цифр влево - 0.0000015) */
```

ОПЕРАЦИИ

```php
<?php

$float = -7.4;

$float++; /* постинкремент - возвращает значение и
увеличивает на 1 */

$float--; /* постдекремент - возвращает значение и уменьшает
на 1 */

++$float; /* преинкремент - увеличивает на 1 и возвращает
значение */

--$float; /* предекремент - уменьшает на 1 и возвращает
значение */

$i = -$float; // отрицание (7.4)

$j = $i + 2.2; // сложение (9.6)

$k = $j - 3.3; // вычитание (6.3)

$l = 1.1 * 9.0; // умножение (9.9)

$m = 1.5 / 2.0; // деление (0.75)

$n = 2.5 ** 2.0; // возведение в степень (6.25)

$o = 3.7 + 10; /* если с числом с плавающей запятой
используется целое число, целое число преобразуется в дробное
(3.7+10.0=13.7) */
```

ПРОБЛЕМЫ С ТОЧНОСТЬЮ ВЫЧИСЛЕНИЙ

- Дробные числа не всегда могут быть точно вычислены из-за особенности их представления в оперативной памяти - например 0.1 + 0.2 не эквивалентно 0.3. Таким образом, дробные числа в PHP напрямую сравнивать не рекомендуется.

ОГРАНИЧЕНИЯ

- константа PHP_FLOAT_MAX позволяет узнать максимальное значение числа с плавающей запятой

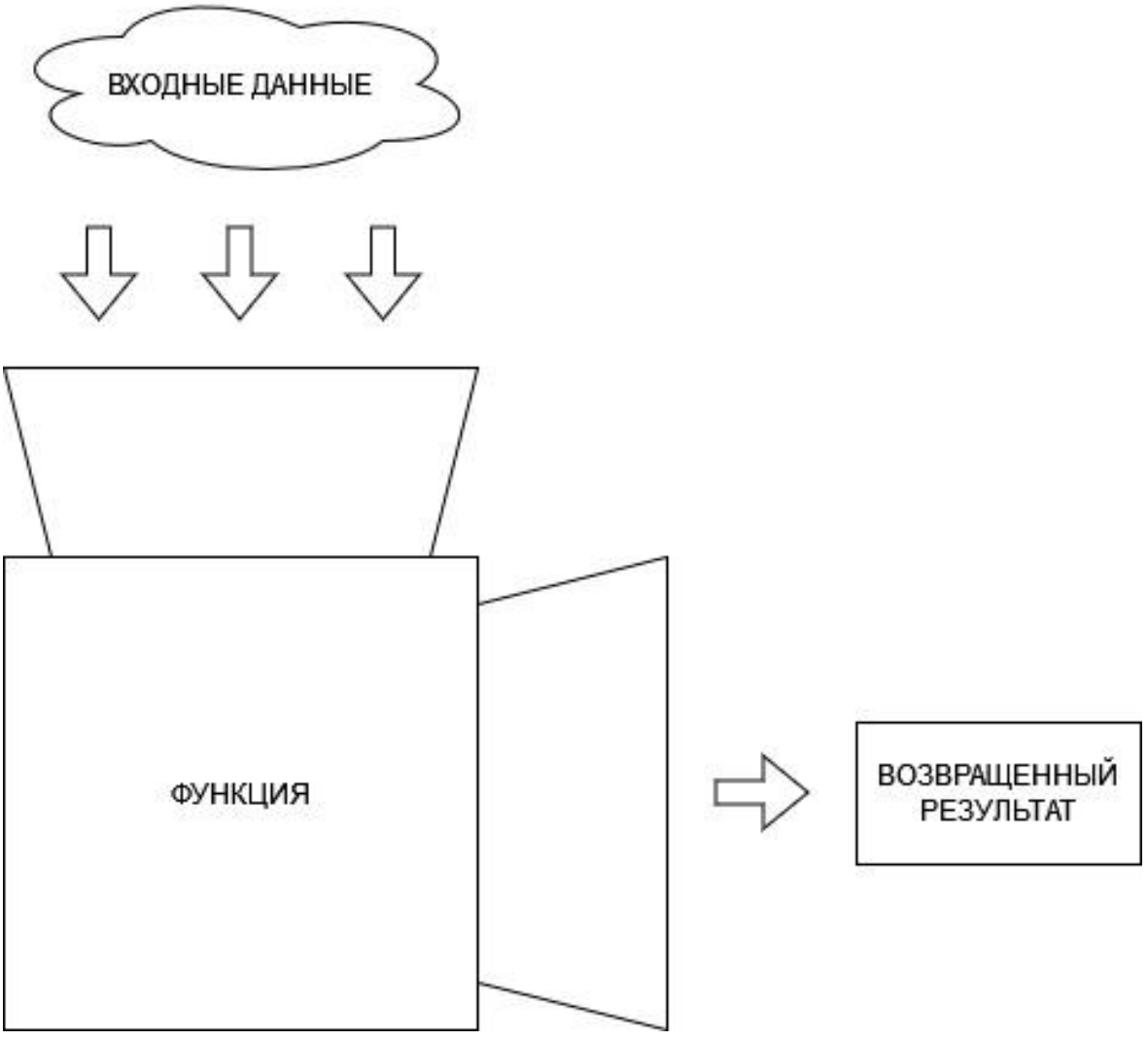
- константа PHP_FLOAT_MIN позволяет узнать минимальное значение числа с плавающей запятой

- при выходе за пределы максимального значения дробное число получит специальное значение INF (бесконечность)

ФУНКЦИИ

ВВЕДЕНИЕ

- часто одна и та же логика (например, логика вычисления суммы числа) необходима в разных частях программы — чтобы не повторяться, в PHP встроена возможность писать функции — блоки кода, которые можно использовать многократно без полного переписывания того, что находится внутри этой самой функции



ПРИМЕРЫ

```php
<?php

/*
     функция может иметь параметры (в данном случае это
     $a и $b) - т.е. данные, которые передаются в функцию, а
     так же возвращаемое значение (return), которое
     подставляется в то место, где вызывается функция
*/
function calculateSum(int $a, int $b): int
{
     $sum = $a + $b; // здесь находится
     return $sum;    // тело функции
}
// значение переменной $resultOfSumFunction будет равно 30
$resultOfSumFunction = calculateSum(10, 20);

// функции могут участвовать в более сложных выражениях
$res = calculateSum(1, 2) + calculateSum(3, 4);
```

```php
<?php

/*
     функция может ничего не возвращать, тогда в
     качестве возвращаемого типа следует указывать void
*/
function printSum(int $a, int $b): void
{
     $sum = $a + $b;
     echo $sum;
}

// возвращено ничего не будет, но на печать будет выведено 70
printSum(30, 40);
```

```php
<?php

// функции могут иметь значение параметров по умолчанию
function multiply(int $a, int $b = 10): int
{
     return $a * $b;
}

/*
     на печать будет выведено 500, т.к. в качестве первого
     параметра было передано 50, а у второго было взято
     значение по умолчанию — 10; если явно передать второй
     параметр, то значение по умолчанию будет проигнорировано
*/
echo multiply(50);
```

```php
<?php

/*
     функции могут быть анонимными
*/
$substraction = function (int $a, int $b): int {
     return $a - $b;
};

// на печать будет выведено 4
echo $substraction(7, 3);
```

ОБЛАСТЬ ВИДИМОСТИ

- переменные внутри тела функции не имеют доступа к переменным вне этой функции:

```php
<?php

/*
     эта переменная $result ничего не знает о переменной с
     таким же названием внутри функции divide
*/
$result = 10;
function divide(int $a, int $b): int
{
     $result = $a / $b;
     return $result;
}

echo divide(10, 2); // на печать будет выведено 5
echo $result; // на печать будет выведено 10
```

- есть особый случай, когда код внутри функции все же может обратиться к какой-либо внешней переменной при помощи модификатора `global`:

```php
<?php

$someVar = 10;
function modifySomeVar(): void
{
     global $someVar; // обращение ко внешней переменной
     $someVar = 20;
}

modifySomeVar();

echo $someVar; // на печать будет выведено 20
```

- существует еще один особый случай, когда функция может изменить значение переменной, которая подается в функцию в качестве параметра — используется оператор &; если такого параметра нет, то значение переменной не изменится:

```php
<?php

$param = 20;
function modifyParam(int &$paramWithOtherName): void
{
     $paramWithOtherName = 100;
}

modifyParam($param);

echo $param; // на печать будет выведено 100
```

ОСОБЕННОСТИ ОБЪЯВЛЕНИЯ

- перед началом исполнения код сканируется на объявление функций — даже если функция объявлена ниже, чем она вызвана — код выполнится успешно:

```php
<?php

echo sumThreeNumbers(10, 20, 30); // выведется 60

// функция объявлена ниже, чем она вызывается!
function sumThreeNumbers(int $a, int $b, int $c): int
{
     return $a + $b + $c;
}
```

ПОРЯДОК ПЕРЕДАЧИ ПАРАМЕТРОВ В ФУНКЦИЮ

- Начиная с PHP 8.0 необязательно передавать параметры функции в том порядке, в котором они были объявлены — отныне можно указать имя параметра, а затем через двоеточие либо переменную, либо необходимое значение:

```
<?php

function divide(float $a, float $b): float
{
     return $a / $b;
}

echo divide(b: 10, a: 20); /* на печать будет выведено 2,
т.к. в качестве делимого (первый параметр $a) было
передано число 20, а в качестве делителя (второй параметр
$b) – число 10 */

echo divide(10, 20); /* стандартный случай - выведется 0.5,
т.к. в данном случае не были указаны имена
переданных параметров и их значения взяты в том порядке, к
котором эти переменные были объявлены */
```

ВСТРОЕННЫЕ ФУНКЦИИ

- кроме возможности создавать свои собственные функции, PHP предоставляет возможность использовать тысячи встроенных функций, например:

```
<?php

$x = 20;

unset($x); // удаляет переменную

round(10.499, 2); /* округление чисел до определенного знака после запятой — будет возвращено 10.5 */

random_int(0, 100); /* возвращает случайное число от переданного минимального числа до заданного максимального числа */

gettype(10); /* возвращает тип переданного значение или переданной переменной — в данном случает будет возвращено integer */

abs(-100); /* возвращает модуль числа (всегда положительное значение или ноль) — в данном случае будет возвращено 100 */
```

ТИПЫ ДАННЫХ 3: СТРОКА (STRING)

ОБЪЯВЛЕНИЕ

```php
<?php

$first = 'Hello, World!'; // строка в одинарных кавычках

$second = "Hello, User!";  // строка в двойных кавычках
```

```php
<?php

$num = 18;

echo "I am $num years old."; /* в двойных кавычках в строку
можно подставлять переменные — выведется I am 18 years old,
кроме того, в двойные кавычки можно подставлять специальные
символы, например, \n (перевод на другую строку) — в
одинарных кавычках он будет выведен как есть, а в двойных
преобразован в перевод на другую строку */
```

```php
<?php

$programmingLanguage = 'PHP';
/*
     heredoc - особый синтаксис объявления строки, когда
     учитываются все отступы, переносы строк и подставляются
     все переменные; в ранних версиях PHP (до 7.3.0)
     закрывающая метка должна была быть в самом начале
строки,
     в современных версиях — может иметь отступ
*/
$str = <<<HEREDOC
     I'm learning
          $programmingLanguage language!
HEREDOC;
```

```php
<?php

$programmingLanguage = 'PHP';
/*
     nowdoc – еще один вариант объявления строки, когда
     учитываются все отступы, переносы строк, но переменные
не подставляются; особенности обработки отступа закрывающей
     метки — такие же, как и в случае с heredoc
*/
$str = <<<'NOWDOC'
     I can write outer variables,
          for example $programmingLanguage ,
               but they will be shown without execution!
NOWDOC;
```

ОПЕРАЦИИ

```
<?php

$str = 'Hello, ' . 'World' . '!'; /* операция соединения двух
или более строк (конкатенация) — получится Hello, World! */
```

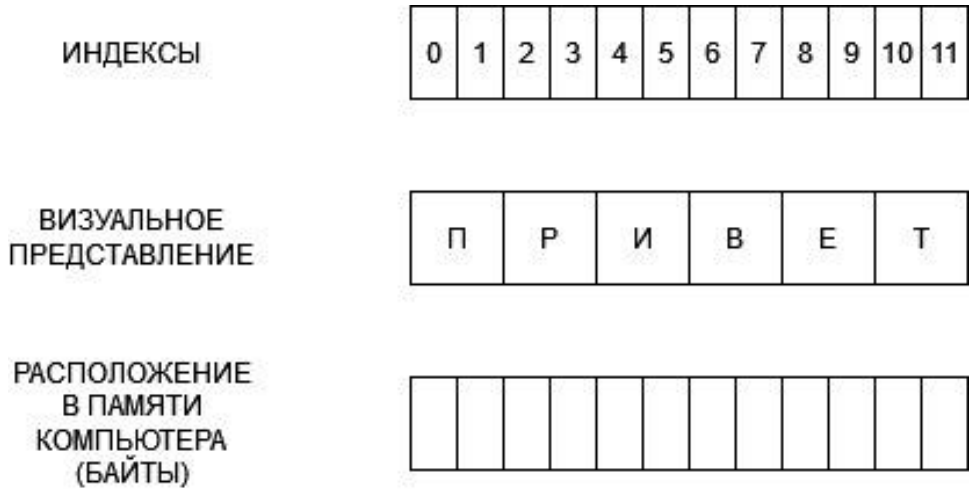
ВЗЯТИЕ ОТДЕЛЬНЫХ СИМВОЛОВ И ПОДСТРОК

- если строка содержит только буквы латинского алфавита или цифры (т.н. однобайтовые символы), то получить отдельные символы из строки можно при помощи квадратных скобок, в которых вписан индекс символа (нумерация индексов строки начинается с 0):

```php
<?php

$str = 'Hello';
echo $str[0]; // выведется H
echo $str[1]; // выведется e

/*
     если мы хотим узнать длину строки с латинскими буквами,
     надо использовать встроенную функцию strlen:
*/
echo strlen($str); // выведется 5
```

- **крайне важно помнить, что строки в PHP являются мультибайтовыми (несколько байт на один символ), поэтому использование операций, определенных в предыдущем пункте, не рекомендуется, если в строке могут быть буквы русского, латышского или других языков с нелатинскими знаками!**

```php
<?php

$str = 'Привет';
echo $str[0]; // выведется 
echo strlen($str); // выведется 12
```



- **самый лучший вариант работы со всеми вариантами строк - использование функций расширения mbstring (но вначале следует убедиться, что это расширение включено в файле php.ini):**

```php
<?php

$str = 'Всё будет работать!';
echo mb_substr($str, 0, 1); // выведется В
echo mb_substr($str, 1, 1); // выведется с
echo mb_strlen($str); // выведется 19
echo mb_substr($str, 4, 5); /* взятие подстроки — выведется
будет */
```

ТИПЫ ДАННЫХ 4: ЛОГИЧЕСКИЙ ТИП (BOOL)

ОБЪЯВЛЕНИЕ

```php
<?php

$bool1 = true; // истина

$bool2 = false; // ложь

$bool3 = 2 + 2 === 4; /* true (истина) - сравнение на
эквивалентность с учетом типа */

$bool4 = 1.0 == 1; /* true (истина) - сравнение на
эквивалентность без учета типа данных */

$bool5 = '1' === 1; /* false (ложь) — выражение, схожее с
предыдущим, но используется сравнение с учетом типа данных */

$bool6 = 1 + 1 != 3; /* true (истина) - сравнение на
отсутствие эквивалентности без учета типа данных */

$bool7 = 1.0 !== 1; /* true (истина) - сравнение на
отсутствие эквивалентности с учетом типа данных */

$bool8 = '5' > 2; // true (истина)

$bool9 = 100 < 3; // false (ложь)

$bool10 = 7 >= 7; // true (истина)

$bool11 = 100 <= 3; // false (ложь)
```

ОПЕРАЦИИ

```php
<?php

$bool = false;

$p = !$bool; // отрицание - true (истина)

$r = $p && true; /* логическое «и» - истина, если оба члена выражения истинны */

$s = $r || false; /* логическое «или» - истина, если   хотя бы один из членов выражения истинен */

$t = true && true || false; /* если в выражении используется как логические «и», так и логические «или», первыми будут вычислены логические «и» */
```

ПРЕОБРАЗОВАНИЯ

- При выводе на печать и конкатенации со строкой true преобразуется в '1', false преобразуется в пустую строку

- логический тип может участвовать в вычислениях с целыми числами и числами с плавающей точкой — true трансформируется в 1 (с целыми числами) и 1.0 (с числами с плавающей точкой), а false - в 0 и 0.0 соответственно

- при проверке на истинность (см. следующую главу) все числа кроме 0 и 0.0 преобразуются в true, также в true преобразуются все строки кроме пустой строки '' и строкового нуля '0'

УСЛОВНЫЕ КОНСТРУКЦИИ

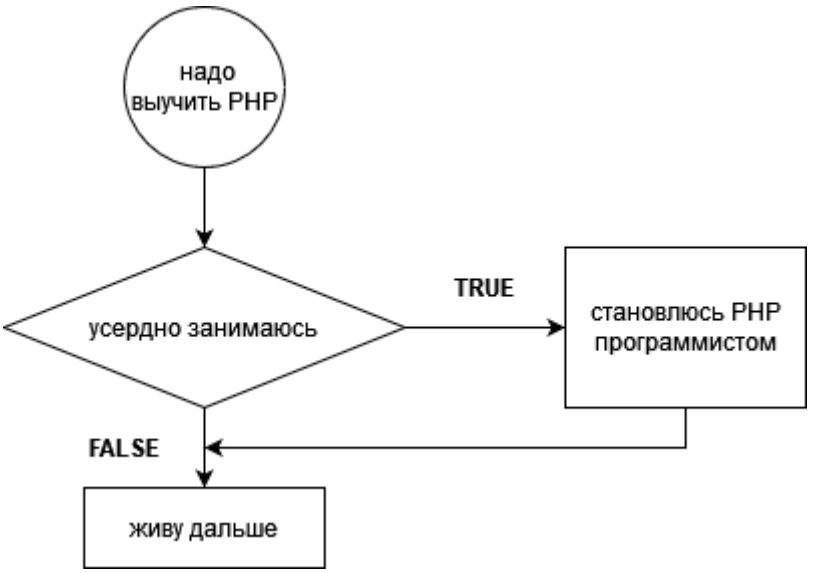
ВВЕДЕНИЕ

- условные конструкции позволяют выполнить те или иные блоки кода в зависимости от неких условий

- условия выражаются при помощи логических типов (если в условия будет передан какой-либо другой тип - он будет преобразован в логический - см. предыдущую главу)

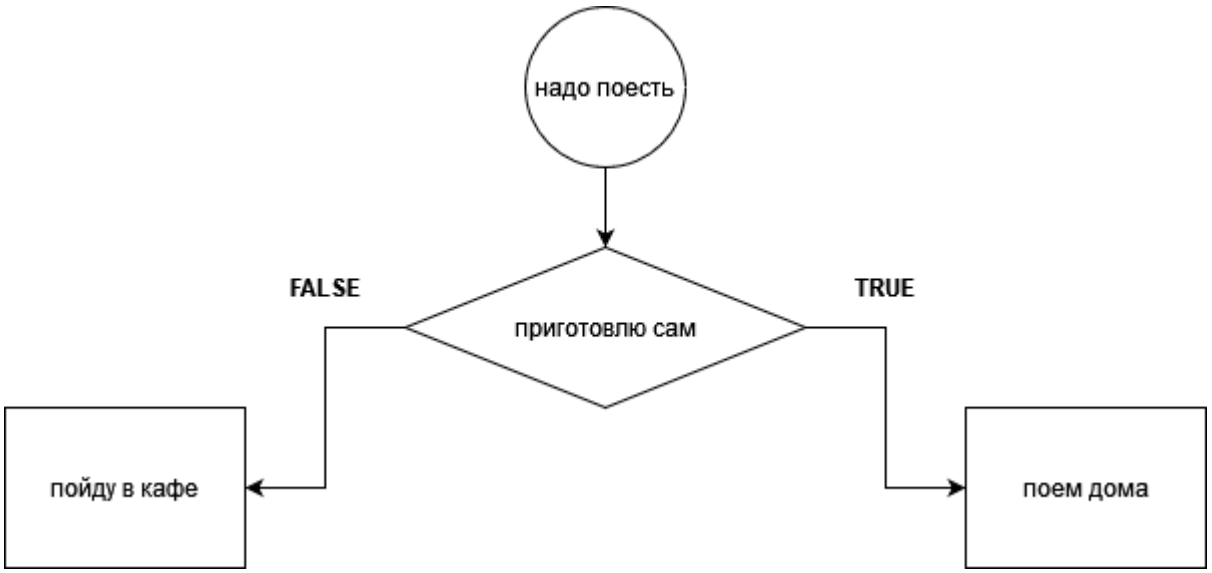
- код внутри блока условной конструкции имеет доступ ко внешнему коду (важное отличие от тела функции)!

ВЫПОЛНЕНИЕ БЛОКА КОДА ПРИ ИСТИННОСТИ УСЛОВИЯ: IF



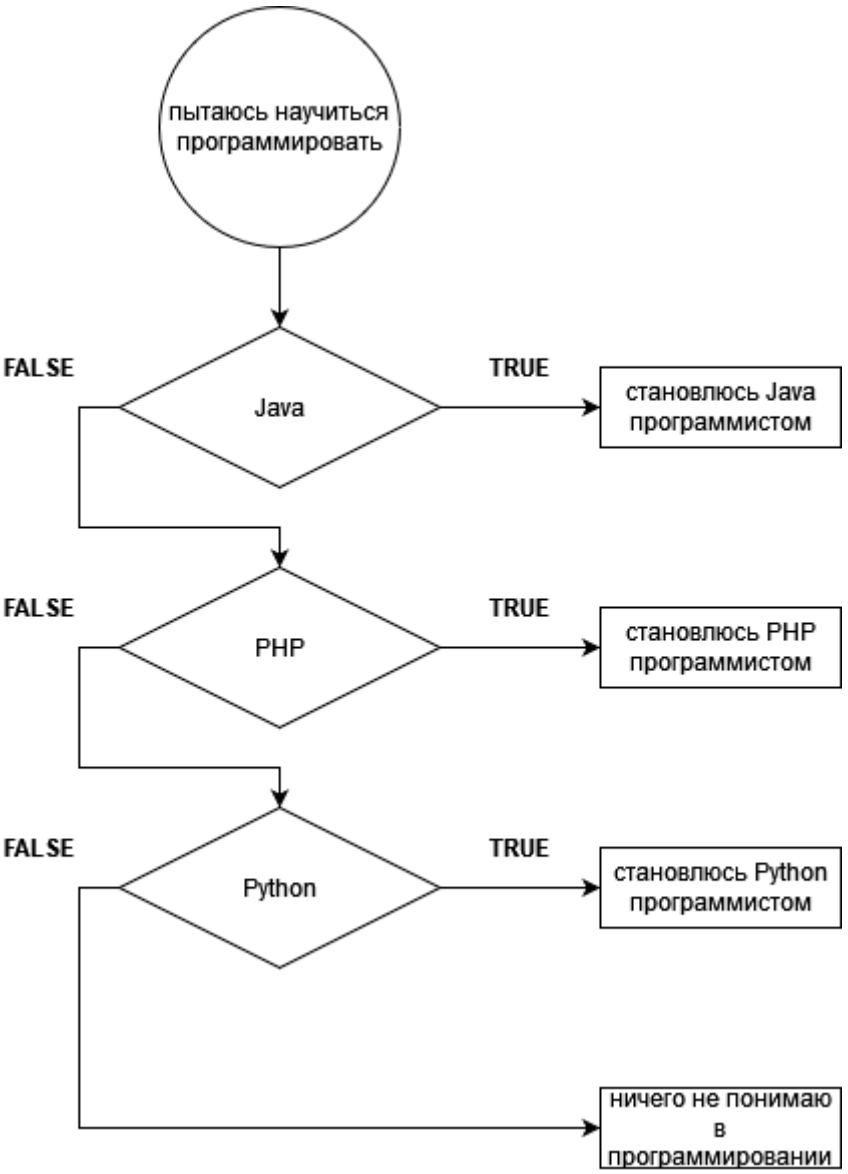

```php
<?php

$tryingHard = true;
if ($tryingHard) { // в скобках нужен логический тип
     echo 'становлюсь PHP программистом';
}
echo 'живу дальше';
```

АЛЬТЕРНАТИВА ПРИ НЕВЫПОЛНЕНИИ УСЛОВИЯ: IF-ELSE




```php
<?php

$cookByHimself = false;
if ($cookByHimself) {
     echo 'поем дома';
} else {
     echo 'пойду в кафе'; // результат
}
```

НЕСКОЛЬКО АЛЬТЕРНАТИВНЫХ УСЛОВИЙ: ELSEIF




```php
<?php

$programmingLanguage = 'PHP';
if ($programmingLanguage === 'Java') {
     echo 'становлюсь Java программистом';
} elseif ($programmingLanguage === 'PHP') {
     echo 'становлюсь PHP программистом'; // результат
} elseif ($programmingLanguage === 'Python') {
     echo 'становлюсь Python программистом';
} else {
     echo 'ничего не понимаю в программировании';
}
```

ТЕРНАРНЫЙ ОПЕРАТОР ?

- если переменной надо назначить одно из двух значений, которые зависят от одного условия (аналог if-else), то можно использовать более короткий тернарный оператор ?

```php
<?php

$age = 33;

$cinemaTicketPrice = $age >= 18 ? 20 : 10;

echo $cinemaTicketPrice; // 20
```

АЛЬТЕРНАТИВА КОНСТРУКЦИИ ELSEIF: ОПЕРАТОР SWITCH

- главное отличие конструкции elseif от оператора switch состоит в том, что elseif позволяет использовать более сложные условия при определении правильного варианта (несколько логических «и» и «или»):

```php
<?php

$day = 4;
switch ($day) {
     /*
          можно писать одновременно несколько условий -
          case 1:
          case 2:
     */
     case 1:
          echo 'Понедельник';
          break;
     case 2:
          echo 'Вторник';
          break;
     case 3:
          echo 'Среда';
          break;
     case 4;
          echo 'Четверг'; // результат
          break;
     case 5:
          echo 'Пятница';
          break;
     case 6:
          echo 'Суббота';
          break;
     case 7;
          echo 'Воскресенье';
          break;
     default: // необязательный блок по умолчанию
          echo 'Неизвестный день недели';
}
```

ТИПЫ ДАННЫХ 5: МАССИВ (ARRAY)

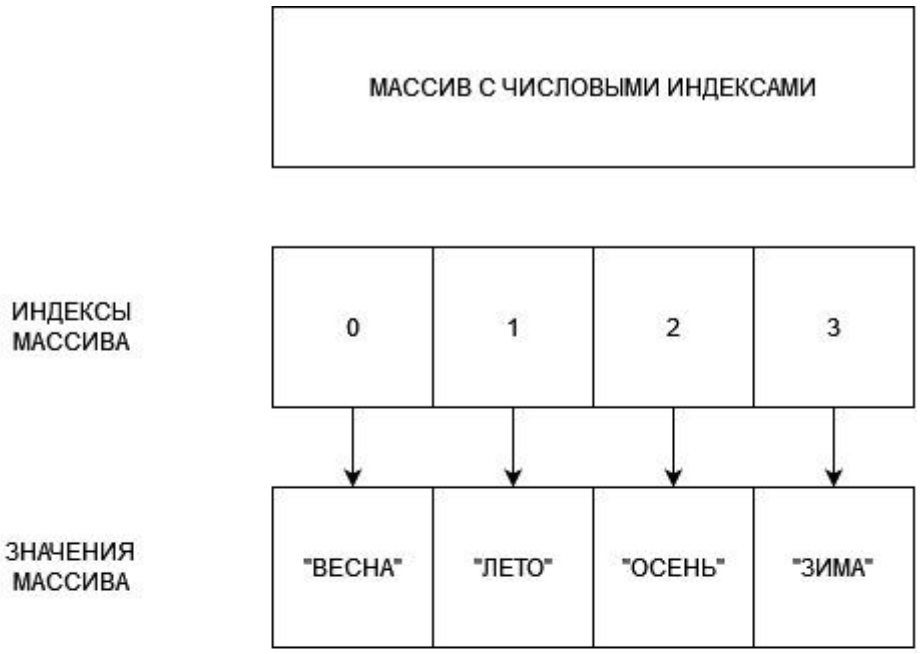
ВВЕДЕНИЕ

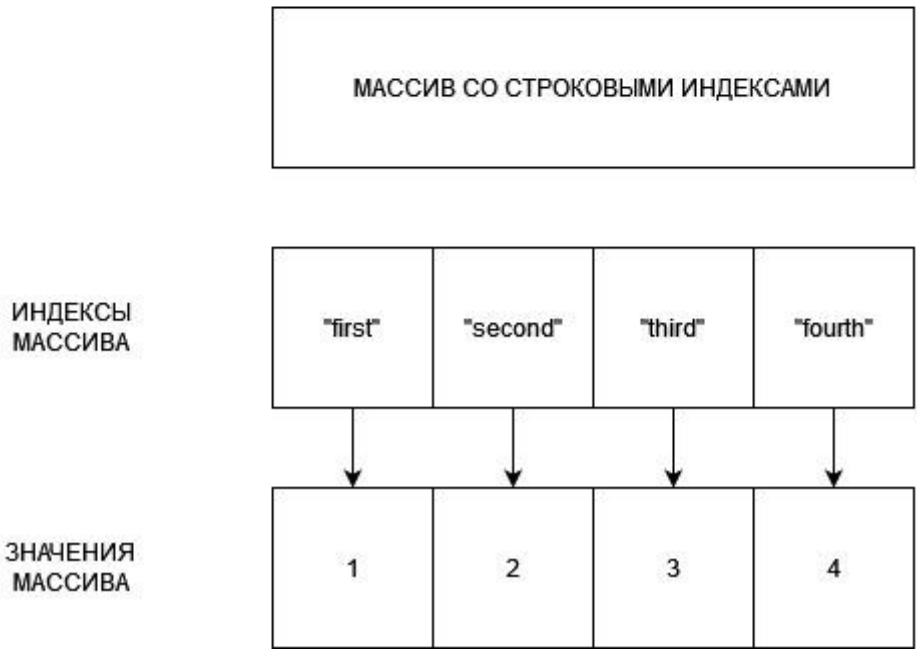
- очень часто необходимо работать не только с одиночными значениями (числа, строки), но и с наборами значений (например, список заказов, список дней недели и т.д.) - для таких целей в PHP принято применять еще один тип данных — массивы

- в массиве можно хранить элементы любого типа — целые числа, дробные числа, логические типы данных, строки и даже другие массивы, причем в одном массиве можно хранить одновременно разные типы данных (т.е. в одном и том же массиве может быть и строка, и число и т.д.)

- для обращения к элементам массива используются ключи или индексы — эти индексы могут быть либо целыми числами, либо строками (массив с индексами-строками также называют ассоциативным массивом); в одном и том же массиве можно использовать одновременно оба вида индексов, но по умолчанию используются числа

- если в массиве в качестве индексов используются числа и эти индексы не указываются явно, то нумерация индексов начинается с нуля


МАССИВ С ЧИСЛОВЫМИ ИНДЕКСАМИ
ИНДЕКСЫ МАССИВА
0
1
2
3
ЗНАЧЕНИЯ МАССИВА
"ВЕСНА"
"ЛЕТО"
"ОСЕНЬ"
"ЗИМА"


МАССИВ СО СТРОКОВЫМИ ИНДЕКСАМИ
ИНДЕКСЫ МАССИВА
"first"
"second"
"third"
"fourth"
ЗНАЧЕНИЯ МАССИВА
1
2
3
4

ОБЪЯВЛЕНИЕ

```php
<?php

$arr = []; /* объявления пустого массива, также очень редко
может использоваться старый вид объявления: $arr = array();
*/

$seasons = ['Весна', 'Лето', 'Осень', 'Зима']; /* объявления
массива с элементами */

echo $seasons[2]; /* обращение к элементу массива по
числовому индексу - выведется Осень, т.к. используется
числовая
индексация по умолчанию (начиная с 0) */

$weekdays = [];

$weekdays[] = 'Понедельник'; /* вставка элемента в массив —
т.к. до этого массив был пустой, элемент получит индекс 0 */
$weekdays[6] = 'Воскресенье'; /* вставка элемента в массив с
числовым индексом - этот элемент получит индекс 6 */

unset($weekdays[6]); /* удаление элемента массива с индексом
6, останется [0 => 'Понедельник'] */
```

```php
<?php

$user = [
     'name' => 'John', 'surname' => 'Smith'
]; // объявление массива со строковыми индексами

$user['age'] = 18; /* вставка элемента в массив со строковым
индексом (ассоциативный массив) */

echo $user['name']; /* обращение к элементу по строковому
индексу — выведется John */

unset($user['age']); /* удаление элемента по строковому
индексу, останется ['name' => 'John', 'surname' => 'Smith']
*/
```

ОПЕРАЦИИ

- в PHP отсутствуют специальные операторы для работы с массивами, но существует множество встроенных функций, которые позволяют взаимодействовать с этим типом данных:

```php
<?php

$years = [2022, 2023, 2024];

echo count($years); /* функция count применяется для
получения размера массива — выведется 3 */

array_push($years, 2025); /* добавление элемента в конец
массива (меняет переданный массив) */

array_unshift($years, 2021); /* добавление элемента в начало
массива (меняет переданный массив) */

array_pop($years); /* удаление последнего элемента массива
(меняет переданный массив) */

array_shift($years); /* удаление первого элемента
массива(меняет переданный массив) */

array_reverse($years); // возвращает массив в обратном
порядке

$a = [1, 2, 3];
$b = [4, 5, 6];
$c = [7, 8, 9];
array_merge($a, $b, $c); /* вернет массив, в котором будут
представлены элементы всех переданных массивов */

$nums = [100, 200, 300];
array_search(100, $nums); /* возвращает индекс переданного
значение в массиве, если значение не найдено, функция вернет
false */

$names = ['John', 'Mary', 'Dave'];
array_keys($names); /* возвращает массив индексов переданного
массива — [0, 1, 2] */
```

ЦИКЛЫ

ВВЕДЕНИЕ

- иногда надо повторить определенные действия множество раз в определенных условиях (например, вывод на печать элементов массива) — именно для это был придуман еще один тип управляющих конструкций — циклы

- существует 4 вида циклов — while, do-while, for и foreach

ЦИКЛ WHILE

- цикл, который перед исполнением каждой итерации проверяет определенное условие — если условие истинно (true), то итерация будет выполнена, если ложно — остановится

```php
<?php

// в процессе выполнение цикла выведется 0123456789
$i = 0;
while ($i < 10) { // в скобках — проверка условия
     echo $i; // тело
     $i++;    // цикла
}
```

```php
<?php

/*
     важно избегать бесконечных циклов — это случится, если
     условие цикла всегда истинно; ниже пример цикла, который
     бесконечно печатает 1
*/
while (true) {
     echo 1;
}
```

```php
<?php

/*
     цикл можно прервать (до его нормального завершения) при
     помощи команды break - в данном случае будет выведено
     только 012345
*/
$j = 0;
while ($j < 10) {
     echo $j;
     $j++;
     if ($j > 5) {
          break; // выход из цикла
     }
}
```

```php
<?php

/*
     некоторые итерации цикла можно пропустить при помощи
     команды continue – в данном случае будут выведены
     цифры от 1 до 9, но 5 будет пропущена
*/
$k = 0;

while ($k < 10) {
     $k++;
     if ($k === 5) {
          continue; // пропуск итерации
     }
     echo $k;
}
```

ЦИКЛ DO-WHILE

- единственное отличие цикла do-while от цикла while заключается в том, что do-while осуществляет проверку на истинность после каждой итерации — таким образом, do-while гарантированно исполнится хотя бы один раз

- как и в случае цикла while - команда continue пропускает итерацию, а команда break принудительно останавливает цикл

```
<?php

/*
     несмотря на то, что условие всегда ложно ($i всегда
     больше нуля или равно нулю) — на печать один раз будет
выведен 0
*/
$i = 0;

do {
     echo $i;
     $i++;
} while ($i < 0); // в скобках — проверка условия
```

```
<?php

/*
     простейшее использование цикла do-while – будет
     выведено 0123456789
*/

$j = 0;

do {
     echo $j;
     $j++;
} while ($j < 10);
```

ЦИКЛ FOR

- цикл, который, как и цикл while, проверяет условие перед каждой итераций, но одновременно дает возможность явно сделать некие действия перед началом цикла, а также явно сделать некоторые действия после каждой итерации

- имеет следующий вид:

```
for (начало; условие; действие после итерации) {
     // тело цикла
}
```

- любой цикл for может быть переписан как while и наоборот

- как и в случае цикла while - команда continue пропускает итерацию, а команда break останавливает цикл

```
<?php

/*
     данный цикл печатает цифры от 0 до 9
     в первой части определяется начальное условие: $i = 0
     во второй части проверяется условие перед каждой
     итерацией - если ложно — цикл прервется: $i < 10
     в третьей части — после каждой итерации увеличивается
     счетчик цикла
*/
for ($i = 0; $i < 10; $i++) {
     echo $i;
}
```

```php
<?php

/*
     особенно удобно использовать цикл for при обходе
     массива с числовыми индексами - в данном случае будет
выведено abc
*/
$arr = ['a', 'b', 'c'];

$ci = count($arr);
for ($i = 0; $i < $ci; $i++) {
     echo $arr[$i];
}
```

```php
<?php

/*
     ни одна из частей инициализации цикла не является
     обязательной — если их проигнорировать, и не
     прописать никаких дополнительных условий, цикл будет
     выполняться бесконечно
*/
for (;;) {
     echo 1;
}
```

ЦИКЛ FOREACH

- цикл, который позволяет перебирать не только массивы с числовыми индексами, но и массивы со строковыми индексами (а также некоторые объекты, но это материал для будущего изучения)

- как и в случае цикла while и for - команда continue пропускает итерацию, а команда break останавливает цикл

- существуют два вида синтаксиса этого цикла

```php
<?php

/*
     цикл foreach только для перебора значений массива, в
     данном случае будет выведено 123
*/
$arr = ['a' => 1, 'b' => 2, 'c' => 3];
foreach ($arr as $val) { // $val – значение массива
     echo $val;
}
```

```php
<?php

/*
     цикл foreach для перебора значений и индексов массива
     $val – значение массива, $key – индекс массива
     в результате будет выведено a – 1 b – 2 c - 3
*/
foreach ($arr as $key => $val) {
     echo $key . ' - ' . $val . ' ';
}
```

```php
<?php

/*
     цикл foreach может перебирать массивы и с числовыми
     индексами — в данном случае будет выведено 10 20 30
*/
$arr2 = [10, 20, 30];
foreach ($arr2 as $val) {
     echo $val . ' ';
}
```

```php
<?php

/*
     при помощи оператора & можно брать значения массива по
     ссылке и менять значение прямо в теле цикла — в данном
     случае значения массива увеличатся в 10 раз
*/
foreach ($arr2 as &$val) {
     $val = $val * 10;
}
unset($val); /* если в цикле была передача по ссылке, то
рекомендуется удалить эту ссылку, т.к. она позволяет
неосознанно менять последнее значение массива! */
```

РАБОТА С ДАННЫМИ СЕРВЕРА: $_GET И $_POST

ВВЕДЕНИЕ

- Интернет основан на обмене данными между компьютерами — существует множество способов и форматов передачи данных, но в контексте языка программирования PHP основным форматом является протокол HTTP (HyperText Transfer Protocol)

- в основе протокола HTTP лежит следующий принцип — инициатор (клиент) формирует в определенном текстовом формате запрос, который после прохождения по сети попадает на сервер, который в свою очередь формирует текстовый ответ клиенту

- в рамках протокола HTTP существует ряд методов передачи данных, но основными являются два — GET (мы говорим серверу, что хотим получить данные по определенному адресу) и POST (мы хотим сохранить какие-либо данные на сервере)

- HTTP запрос можно условно разделить на три части:
  1. стартовая строка — метод запроса, путь запрашиваемой страницы (к которому можно прикрепить дополнительные значения), а также версия протокола

  2. заголовки запроса — уточняющие запрос параметры (например, необходимый формат данных)

  3. тело запроса — данные, которые передаются с запросом (для метода GET не применяется)

- HTTP ответ также можно разделить на три части:
  1. стартовая строка — содержит версию HTTP протокола и кодовый статус ответа

  2. заголовки - уточняющие ответ параметры (например, отправляемый формат данных)

  3. тело ответа — данные, которые запрашивал клиент (страница сайта в виде HTML) - может отсутствовать

- в PHP для определения типа запроса следует обращаться к встроенному глобальному массиву $_SERVER по ключу REQUEST_METHOD, например: `echo $_SERVER['REQUEST_METHOD'];`

GET ЗАПРОС И ЕГО ОБРАБОТКА

- GET запрос — простейший вид HTTP запроса, часть которого можно увидеть даже в адресной строке браузера:

```
https://example.org/index.php?name=John&surname=Smith
```

- в данном случае браузер обращается к файлу сервера по пути /index.php и передает две уточняющих переменных — name (со значением John) и surname (со значением Smith); далее представлен полный пример GET запроса:

```
GET /index.php?name=John&surname=Smith HTTP/1.1
Host: www.example.com
Accept: text/html
Accept-Language: ru
User-Agent: Mozilla/5.0 ...
```

- пример ответа на GET запрос:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 12345

<!DOCTYPE html>
<html>
...
</html>
```

- для получения данных GET запроса в PHP доступен глобальный массив $_GET – массив является ассоциативным, т.е. индексы этого массива — названия переданных переменных, а значения массива — значения переменных; если мы ожидаем запрос из предыдущего примера, то данные получают следующим образом:

```
<?php

/*
    пример простейшего запроса с параметрами:
    /index.php?name=John&surname=Smith
*/

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
     if (isset($_GET['name']) && is_scalar($_GET['name'])) {
          $name = $_GET['name'];
          echo 'Name:', strip_tags($name);
     }

     echo '<br>';

     if (isset($_GET['surname']) &&
         is_scalar($_GET['surname'])
     ) {
          $surname = $_GET['surname'];
          echo 'Surname:', strip_tags($surname);
     }
}
```

- в рамках GET запроса можно передавать массивы как с числовыми индексами, та и со строковыми:

```
<?php

/*
    пример запроса с числовыми индексами:
    /index.php?names[]=John&names[]=Mary
*/

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
     if (isset($_GET['names']) && is_array($_GET['names'])) {
          print_r($_GET['names']);
     }
}
```

```php
<?php

/*
    пример запроса со строковыми индексами:
    /index.php?nums[a]=1&nums[b]=2
*/

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
     if (isset($_GET['nums']) && is_array($_GET['nums'])) {
          $nums = $_GET['nums'];

          if (isset($nums['a']) && is_scalar($nums['a'])) {
               echo 'a:', strip_tags($nums['a']);
          }

          echo '<br>';

          if (isset($nums['b']) && is_scalar($nums['b'])) {
               echo 'b:', strip_tags($nums['b']);
          }
     }
}
```

- GET запрос можно отправить не только через адресную строку браузера, но и через html-формы, которые располагаются прямо на странице сайте, например:

```
<?php

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
     if (isset($_GET['title']) &&
         is_scalar($_GET['title'])
     ) {
          $title = $_GET['title'];
          echo 'Title:', strip_tags($title);
     }

     echo '<br>';

     if (isset($_GET['body']) &&
         is_scalar($_GET['body'])
     ) {
          $body = $_GET['body'];
          echo 'Body:', strip_tags($body);
     }
}
?>

<form action="/index.php" method="GET">
     Title: <input type="text" name="title">
     <br><br>
     Body: <textarea name="body"></textarea>
     <br><br>
     <button type="submit">Submit</button>
</form>
```

POST ЗАПРОС И ЕГО ОБРАБОТКА

- Основная задача POST запроса — передача данных на сервер для последующего их сохранения

- в отличие от GET запроса данные POST запроса не видны в адресной строке браузера и отправляются на сервер посредством html-форм (есть и другие способы отправки POST запроса, но они выходят за рамки этого урока)

- POST запрос содержит т.н. тело, куда как раз и записываются данные — чтобы прочитать эти данные, POST запрос имеет заголовок Content-Length с размером (в байтах) данного тела:

```
POST /index.php HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Accept-Language: ru
User-Agent: Mozilla/5.0 ...
Content-Length: 19

login=John&pwd=1234
```

- в рамках PHP обработка POST запроса идентична обработке GET запроса – существует массив $_POST, из которого можно получать данные либо по названию переданных переменных, либо по числовым индексам, либо по строковым индексам

```
<?php

// получение POST данных (из переданных переменных)
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_POST['firstname']) &&
        is_scalar($_POST['firstname'])
    ) {
          $firstname = $_POST['firstname'];
          echo 'Firstname:', strip_tags($firstname);
    }

    echo '<br>';

    if (isset($_POST['lastname']) &&
        is_scalar($_POST['lastname'])
    ) {
        $lastname = $_POST['lastname'];
        echo 'Lastname:', strip_tags($lastname);
    }
}
?>

<form action="/index.php" method="POST">
     Firstname: <input type="text" name="firstname">
     <br><br>
     Lastname: <input type="text" name="lastname">
     <br><br>
     <button type="submit">Submit</button>
</form>
```

```php
<?php

// получение POST данных (массив с числовыми индексами)
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
     if (isset($_POST['products']) &&
         is_array($_POST['products'])
     ) {
        $products = $_POST['products'];
        print_r($products);
     }
}
?>

<form action="/index.php" method="POST">
     1st product: <input type="text" name="products[]">
     <br><br>
     2nd product: <input type="text" name="products[]">
     <br><br>
     <button type="submit">Submit</button>
</form>
```

```php
<?php

// получение POST данных (массива со строковыми индексами)
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
     if (isset($_POST['address']) &&
         is_array($_POST['address'])
     ) {
          $address = $_POST['address'];

          if (isset($address['city']) &&
              is_scalar($address['city'])
          ) {
               echo 'City:', strip_tags($address['city']);
          }

          echo '<br>';

          if (isset($address['street']) &&
              is_scalar($address['street'])
          ) {
              echo 'Street:', strip_tags($address['street']);
          }
     }
}
?>

<form action="/index.php" method="POST">
     City: <input type="text" name="address[city]">
     <br><br>
     Street: <input type="text" name="address[street]">
     <br><br>
     <button type="submit">Submit</button>
</form>
```